Contenido

2.

Componentes fundamentales App Android

INTRODUCCIÓN

En este tema estudiaremos los componentes de una aplicación Android y sus respectivas funciones.

Lo primero que haremos será comprender que es el contexto de una aplicación y como permite relacionar los componentes.

Luego veremos el concepto de Actividades, Intents, Servicios, BroadCast Receivers y Content Providers.

Para no perdernos en el mare magnum de API's, paquetes y clases que son ofrecidas por el SDK Android ¿qué podemos hacer? Pues bien, antes de escribir ninguna línea de código hay que empezar a comprender alguna de estas API's, fundamentalmente las que vamos a utilizar más frecuentemente.

Para cualquier aplicación Android, en general, será necesario construir una ventana en la que se irán añadiendo componentes gráficos, donde se irán recibiendo entradas del usuario, unido a la posibilidad de ejecutar, por ejemplo, audio.

¿Qué necesitaremos? comenzar la casa por los cimientos y no por el tejado.

En una aplicación Android los cimientos se encuentran en el archivo *manifestFile*. Es aquí donde se integran todos los componentes de la aplicación, algunos de ellos están presentes siempre, sea la aplicación que sea.

Todos estos componentes los desarrollaremos con mayor profundidad en temas venideros.

CONTEXTO DE UNA APLICACIÓN ANDROID

El contexto de una aplicación es una interfaz entre la aplicación y el sistema operativo, la cual describe la información que representa tu aplicación dentro del ambiente del sistema operativo.

También permite acceder a los recursos de la aplicación y coordinar el funcionamiento de los bloques de la aplicación.

El contexto representa toda la meta-información sobre las relaciones que tiene la aplicación con otras aplicaciones o el sistema y podemos implementarlo a través de la clase abstracta Context.

Según la RAE contexto puede definirse como *"Entorno físico o de situación, ya sea político, histórico, cultural o de cualquier otra índole, en el cual se considera un hecho"*.

En Android nos encontramos con los siguientes tres diferentes tipos de contextos:

- Aplicación (Como es lógico, al cubrir todo el ciclo de vida de la aplicación desde que la arrancamos hasta que muere, cada aplicación tiene un único contexto de aplicación; además, este contexto engloba a todos los demás). Se puede acceder desde una Activity o un Service con getApplication(), o desde cualquiera que herede de Context con getApplicationContext().

- Activity o Service

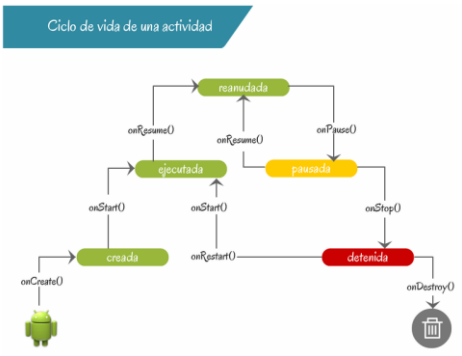
Un Context vive tanto como el elemento al que pertenece, por lo que depende del ciclo de vida. Así, un Context de una Activity vivirá el mismo tiempo que esta, y el Context de la aplicación vive hasta que se termina por completo la aplicación (hasta que muere, no hasta que se pausa).

ACTIVITY

En primer lugar, *Activitys,* que son componentes visibles, con una *Interface de Usuario* que permite interactuar con ellas.

La mayoría de las aplicaciones permiten la ejecución de varias acciones a través de la existencia de una o más pantallas. Por ejemplo, piénsese en una aplicación de mensajes de texto. En ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra tercera puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas puede estar representada a través de un componente *Activity,* de forma que navegar de una ventana a otra implica lanzar una actividad o dormir otra. Android permite controlar por completo el ciclo de vida de los componentes *Activity.*

Los estados por los cuales puede transcurrir una aplicación son los siguientes: Creación, Ejecución, Reanudación, Pausa, Parada y Destrucción. A la relación entre ellos se le llama Ciclo de vida de una actividad.



El ciclo de vida de una Activity nos describe los estados y las transiciones entre estados que una determinada Activity puede tener:

- Creación: Una actividad se ha creado cuando su estructura se encuentra en memoria, pero esta no es visible aun. Cuando el usuario presiona sobre el icono de la aplicación

en su dispositivo, el método onCreate() es ejecutado inmediatamente para cargar el layout de la actividad principal en memoria.

- Ejecución-Reanudación: Después de haber sido cargada la actividad se ejecutan en secuencia a el método onStart() y onResume(). Aunque onStart() hace visible la actividad, es onResume() quien le transfiere el foco para que interactúe con el usuario.
- Pausa: Una actividad está en pausa cuando se encuentra en la pantalla parcialmente visible. Un ejemplo sería cuando se abren diálogos que toman el foco superponiéndose a la actividad. El método llamado para la transición hacia la pausa es onPause().
- Detención: Una actividad está detenida cuando no es visible en la pantalla, pero aún se encuentra en memoria y en cualquier momento puede ser reanudada. Cuando una aplicación es enviada a segundo plano se ejecuta el método onStop(). Al reanudar la actividad, se pasa por el método onRestart() hasta llegar a el estado de ejecución y luego al de reanudación.
- Destrucción: Cuando la actividad ya no existe en memoria se encuentra en estado de destrucción. Antes de pasar a destruir la aplicación se ejecuta el método onDestroy(). Es común que la mayoría de actividades no implementen este método, a menos que deban destruir procesos (como servicios) en segundo plano.

Aunque tu aplicación puede tener varias actividades en su estructura, se debe definir una actividad principal. Para hacerlo se debe especificar en tu archivo Android Manifest un componente <actitivy>, con un componente hijo <intent-filter>. Dentro de este componente declararas dos nuevos componentes, <action> y <category>. Al primero le asignaras el elemento enumerado MAIN y al segundo el elemento enumerado LAUNCHER.

Probar el código del ciclo de vida de una actividad. EjercicioResueltoCicloVida

SERVICIES

Un servicio es una entidad que ejecuta instrucciones en segundo plano sin que el usuario lo note en la interfaz. Son muy utilizados para realizar acciones de larga duración mientras las actividades muestran otro tipo de información. Por ejemplo guardar la información en la base de datos, escuchar música mientras se ejecuta la aplicación, administrar conexiones de red, etc.

Los servicios también tienen un ciclo de vida como las actividades, pero este es más corto. Solo se comprende de los estados de Creación, Ejecución y Destrucción.

INTENTS

Un Intent es un mensaje que se envía de un componente a otro, o entre una aplicación a otra para comunicarse.

Así mismo podemos crear comunicaciones entre las actividades de nuestra aplicación para compartir algunos datos.

Los Intents pueden ser implícitos o explícitos. Son explícitos cuando ejecutamos un componente en específico. Son implícitos cuando se entrega una referencia genérica sobre

alguna condición, y aquellas entidades que cumplan ese requisito serán presentadas como candidatas para la tarea.

Un ejemplo de Intent implícito sería cuando deseas compartir un sitio web en alguna red social. Para ello Android nos ofrece en un pequeño dialogo de lista para que elijamos entre cuales de todas las aplicaciones sociales deseamos elegir para compartir. No especificamos que aplicación queríamos, simplemente se mostraron todas las aplicaciones que podrían responder a ese tipo de acción.

Un Intent explicito se da cuando invocamos una actividad con un tipo de clase especifico. Por ejemplo, supón que dentro de nuestra aplicación tenemos dos actividades llamadas A y B. Si decidimos iniciar la actividad B desde la actividad A usaríamos un Intent explicito debido a que "B" es un tipo definido.

CONTENT PROVIDER

Con el componente Content Provider, cualquier aplicación en Android puede almacenar datos en un fichero, en una base de datos SQLite o en cualquier otro formato que considere.

Además, estos datos pueden ser compartidos entre distintas aplicaciones. Una clase que implemente el componente Content Provider contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación.

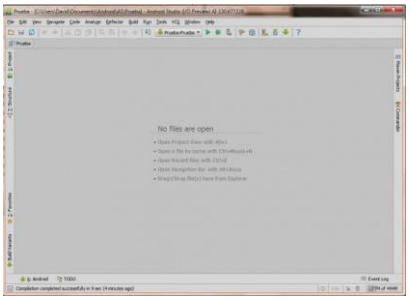
Por defecto, el API de Android trae consigo Content Providers predefinidos para intercambiar información de audio, vídeo, imágenes, e información personal. Pero si en algún momento deseas intercambiar información con una estructura personalizada debes crear tu propia subclase heredada de la clase ContentProvider.

BROADCAST RECEIVES

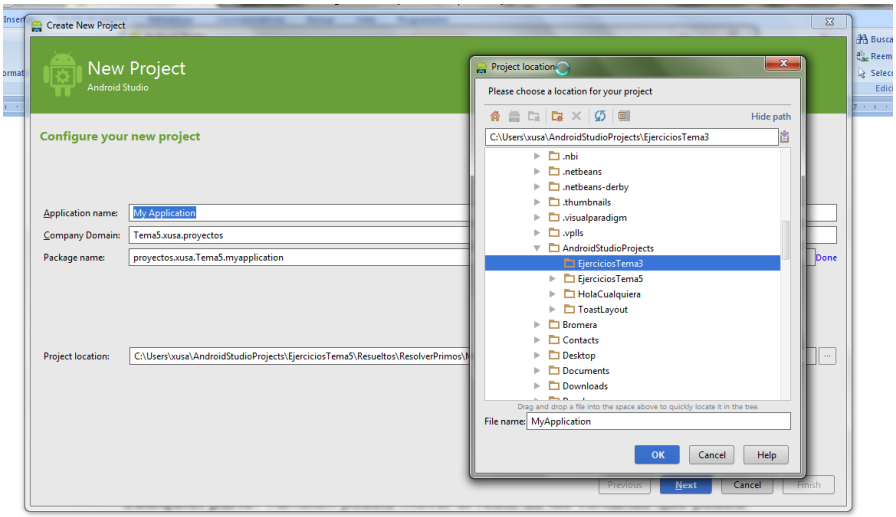
Finalmente, el último componente clave son los Broadcast Receives, o receptores de emisión, que reaccionan a intents específicos pudiendo a su vez ejecutar una acción o iniciando una activity o pueden devolver otro intent al sistema para que siga la ejecución. Realmente este tipo de componentes son capaces de procesar los mensajes del sistema operativo.

CREACIÓN PRIMER PROYECTO ANDROID

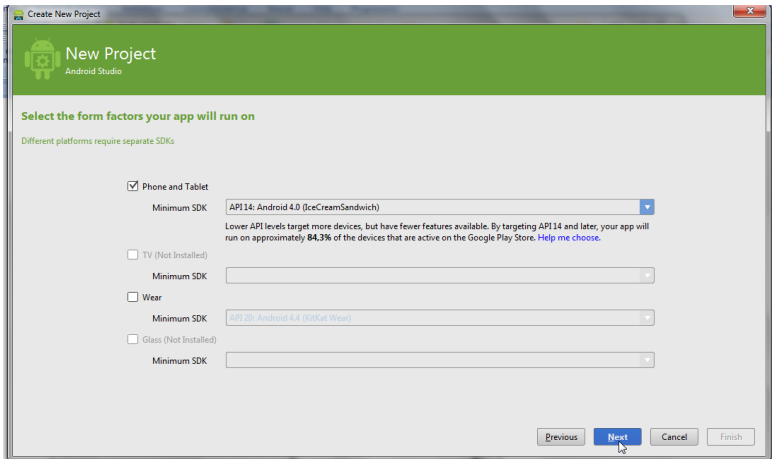
Vamos a realizar nuestro primer ejemplo en Android, nos servirá básicamente para conocer el entorno Android Studio y las carpetas de las que consta un proyecto Android con este entorno. Una vez instalada la aplicación, al abrir el entorno nos encontraremos con una ventana como la que sigue:



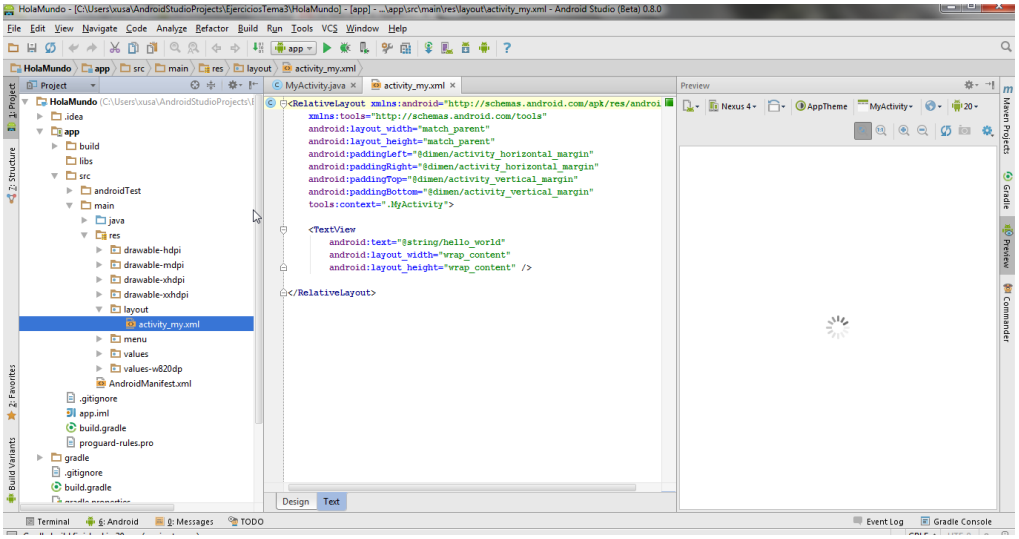
De momento esta ventana está vacía. Pero como vemos en el centro de la ventana, para abrir el directorio de archivos de la aplicación debemos **pulsar File→ New Proyect ó (Alt+1).** A partir de aquí deberemos seleccionar el nombre de paquete que vamos a dar a nuestra aplicación, la localización y el nombre de la aplicación. **Es mejor no editar el nombre de paquete, se generará solo al modificar el nombre de dominio y el nombre de aplicación**. En **Project location** podemos seleccionar donde queremos dejar nuestra aplicación.



La siguiente ventana, permitirá seleccionar el SDK mínimo con el que queráis que corra vuestra aplicación. El máximo no hace falta seleccionarlo porque el proyecto cogerá el SDK más actualizado que tengas instalado. Junto al selector del sdk, podéis ver una explicación y **un enlace** en el que se pueden ver las estadísticas actualizadas del porcentaje de móviles que podrán correr vuestra aplicación, según el mínimo SDK seleccionado.



Las siguientes ventanas las podemos dejar por defecto, o si se quiere se puede cambiar el nombre de la actividad (Activity) y al *layout principal*. A partir de aquí deberemos esperar un poco hasta que se genere nuestro proyecto y aparezca una ventana parecida a la siguiente:



Directorios De Un Proyecto En Android Studio

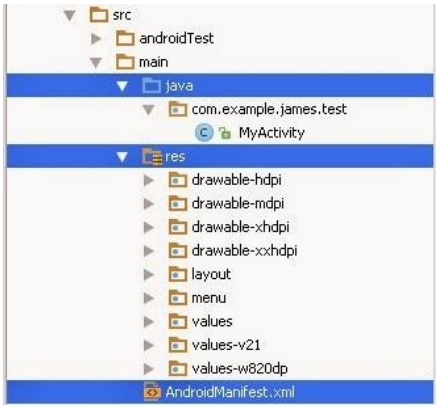
Una vez llegados aquí, se habrá abierto e**l árbol de carpetas de la aplicación a la izquierda** (aunque esta ventana es movible. Por si os resulta más útil en otro lugar de la ventana de Android Studio, sabed que **podéis moverla pulsando y arrastrando donde pone "1.- Project" a cualquier parte**. También podéis mover el resto de las ventanas que están abiertas.

En el árbol de directorios tenéis varias formas de listar las carpetas y archivos de la aplicación. Las dos principales son "Project", y "Package", pero hay otras que podéis usar. Por ejemplo, "Problems" mostrará sólo esos archivos que tienen errores. Algo muy útil. Podéis cambiar entre ellas pulsando sobre la fecha negra en la parte superior de la ventana "1.- Project".

La carpeta app es la que contiene todo lo relacionado con tu proyecto, es la carpeta que nos interesa por el momento y donde incluiremos los archivos necesarios para que nuestra aplicación sea empaquetada. Si despliegas su contenido veras tres carpetas: build, libs y src. Por ahora ignoraremos los demás archivos.

Normalmente la mayor parte del tiempo y fuerzas las usaremos en la carpeta src(Source).

Dentro de ella se ubica la carpeta main, la cual contiene todos los **archivos fuente Java** para nuestra aplicación. La carpeta res (Resources) que contiene los recursos del proyecto (iconos, sonido, diseños, etc.) y el archivo AndroidManifest.xml (ahora profundizaremos en él).



La Activity (o Activitys) que hemos creado se encuentran **en "Nombre de vuestra App">src>main>java>"Paquete de vuestra App".**

Algunos de los otros archivos imprescindibles de toda aplicación Android se encuentran en la carpeta **"Nombre de vuestra App">src>main>res. Una vez dentro podéis ver entre otros, "layout" o "menu".**

¿Qué hay dentro de la carpeta Layout?

En la carpeta layout encontrarás los archivos de diseño de todas tus actividades. En mi caso existe el archivo activiy_my.xml. Este archivo representa el diseño de la interfaz de mi actividad principal. En él se establecerán todos los widgets que vaya a agregar a la actividad.

Construir la interfaz a través de nodos XML es mucho más sencillo que la creación a través de código Java. Adicionalmente **Android Studio** nos ha dotado de un panel de diseño estilo **Drag and Drop**, lo cual es una bendición para los desarrolladores, ya que facilita demasiado la creación de una interfaz de usuario.

Este archivo de diseño comienza con un nodo raíz llamado <RelativeLayout>. Un Layout es el contenedor principal que define el orden y secuencia en que se organizarán los widgets en nuestra actividad. Existen varios tipos de **Layouts,** como por ejemplo el LinearLayout, GridLayout, FrameLayout, etc.

Android Studio crea por defecto un RelativeLayout porque permite crear un grupo de componentes con ubicaciones relativas. Quiere decir que se ubicaran por referencias y no por valores absolutos. Si abres el archivo de diseño de tu actividad verás algo como esto:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MyActivity">
   <TextView
      android:text="@string/hello_world"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</RelativeLayout>
```

Más adelante profundizaremos sobre los elementos de diseño tipo Layouts.

La Carpeta Drawable En Android Studio

Si has sido curioso habrás podido encontrar una serie de carpetas que comienzan por el cualificador drawable. Estas carpetas se relacionan directamente con el tipo de **densidad de pantalla** donde se ejecutará nuestra aplicación.

La mayoría de dispositivos móviles actuales tienen uno de estos 4 tipos de densidades:

- **Mediun Dots per Inch(mdpi)**: Este tipo de pantallas tienen una densidad de 160 puntos por pulgada.
- **High Dots per Inch(hdpi)**: En esta clasificación encontraremos teléfonos cuya resolución es de 240 puntos por pulgada.
- **Extra high dots per inch(xhdpi)**: Resoluciones mayores a 340 puntos por pulgada
- **Extra Extra high dots per inch(xxhdpi)**: Rangos de resoluciones mayores a 480 puntos por pulgada.

Miremos una pequeña imagen ilustrativa:

La imagen muestra algunos ejemplos de dispositivos móviles cuya densidad se encuentra dentro de los rangos establecidos. Al final podemos ver una categoría extra llamada **xxxhdpi**. Esta denominación describe a la densidad de televisores de última generación que ejecutan Android.

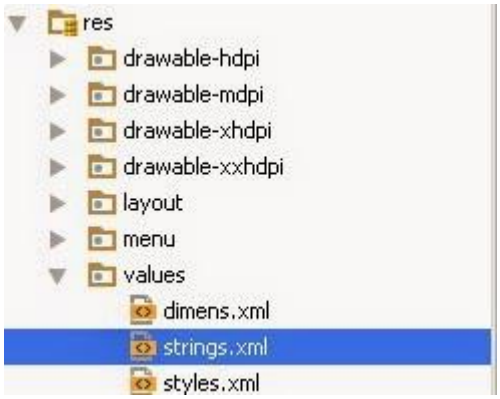
Sería excelente que definieras todos tus recursos gráficos en los distintos tipos de densidades. Esta diversidad permitirá mayor compatibilidad con distintos dispositivos móviles.

¿Qué Utilidad Tiene El Archivo Strings.Xml?

Dentro de la carpeta ”res” encontraremos todos aquellos recursos para nuestra aplicación. Esta práctica de excluir los atributos de la aplicación a través de archivos externos, permite reducir la complejidad de diseño en las interfaces.

Uno de los recursos más relevantes es el archivo strings.xml que se encuentra dentro de la subcarpeta values. Este fichero almacena todas las cadenas que se muestran en los **widgets** (controles, formas, botones, vistas, etc.) de nuestras actividades.

Por ejemplo, si tuvieses un botón cuyo título es *“Presiona aquí”*, es recomendable incluir dicha cadena en tu archivo strings.xml.



Para declarar nuestro recurso de strings usaremos el nodo raíz <resources>. Para declarar las cadenas usaremos la etiqueta <string> y estableceremos el atributo name como identificador. Dentro de esta etiqueta pondremos el texto que se visualizará en el **componente de interfaz**. Esta declaración es similar al uso de la etiqueta <h1>Texto</h1> en **HTML**.

Si abres el archivo, verás que se encuentran tres nodos del tipo <string> : “app_name”, “action_settings” y “hello_world”.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Test</string>
   <string name="hello_world">Hello world!</string>
   <string name="action_settings">Settings</string>
</resources>
```

La etiqueta app_name contiene el nombre de la aplicación. En este caso es “Test”, action_settings se refiere al título del menú acciones y hello_world es el string para nuestro mensaje en pantalla la actividad.

El archivo strings.xml es muy útil para los desarrolladores. Una de sus grandes utilidades es facilitar el uso de **múltiples idiomas** en tu aplicación. Esto se debe a que puedes externalizar

las cadenas del código java y seleccionar la versión del archivo strings.xml con el lenguaje necesitado.

Vamos a definir nuestra aplicación para poder configurarla en tres idiomas: español, inglés y francés, para una aplicación multi-idioma sería necesario incluir recursos de tipo cadena (string.xml) para cada idioma que tenga la aplicación, para ello crearíamos tres versiones del fichero strings.xml y lo guardaríamos en los siguientes tres directorios:

```
res/values/strings.xml
res/values-es/strings.xml
res/values-fr/strings.xml
```

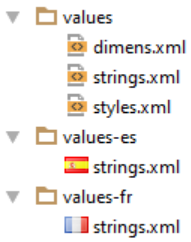
Cada uno de ellos con el siguiente contenido:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorldApp</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HolaMundoApp</string>
    <string name="hello_world">¡Hola Mundo!</string>
    <string name="action_settings">Ajustes</string>
</resources>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BonjourMondeApp</string>
    <string name="hello_world">Bonjour Monde!</string>
    <string name="action_settings">Regláges</string>
</resources>
```

Habremos generado un árbol de directorios como el siguiente:



Al ejecutar la aplicación y dependiendo de la configuración de idioma del dispositivo la aplicación tendrá los mensajes en un idioma u otro.

Personalmente te recomiendo que uses este archivo siempre, incluye todo tu texto y no dejes nada por fuera, tu aplicación te lo agradecerá en el futuro.

¿QUÉ ES El ANDROID MANIFEST?

Es un **archivo XML** que contiene nodos descriptivos sobre las características de una aplicación Android. Características como los **building blocks** existentes, la **versión de SDK** usada, los permisos necesarios para ejecutar algunos servicios, activitys y sevicios de la aplicación y muchas más. En pocas palabras el Android Manifest muestra una descripción de toda nuestra aplicación.

Su configuración puede realizarse a través de una interfaz gráfica, pero es recomendable conocer la sintaxis ya que en muchas ocasiones será más fácil y rápido hacerlo desde el propio xml.

Si abres el archivo en el editor verás un código similar a este:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.TUPAQUETE.test" >

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MyActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Todas las aplicaciones deben contener este archivo por convención. El nombre debe permanecer intacto, ya que se usa como referencia para el **parsing** de nuestra aplicación. El **nodo raíz** de este documento se representa con la etiqueta <manifest> y por obligación debe contener un hijo de tipo <application>.

En el código anterior podemos observar que manifest posee dos atributos, xmlsn:android y package. El primero no debemos cambiarlo nunca, ya que es el **namespace** del archivo.

El atributo package indica el nombre del **paquete Java** que soporta a nuestra aplicación. El nombre del paquete debe ser único y un diferenciador a largo plazo.

La etiqueta <application> representa como estará construida nuestra aplicación. Dentro de ella definiremos nodos referentes a las **actividades** que contiene, las **librerías incluidas**, los **Intents**, **Providers**, y demás componentes.

Etiqueta aplication

Algunos atributos de la etiqueta <application> son:

- allowBackup: Este atributo puede tomar los valores de true o false. Indica si la aplicación será persistente al cerrar nuestro **AVD**.

- icon: Indica donde está ubicado el icono que se usará en la aplicación. Debemos indicar la ruta y el nombre del archivo que representa el icono. En este caso apuntamos a las carpetas drawable donde se encuentra ic_launcher.png. Icono por defecto que nos proporciona **Android Studio**. Se mostrará en el Android Market y en el lanzador de la aplicación de nuestro dispositivo. También será el icono por defecto para todas las activities que definamos dentro de la etiqueta.

- label: Es el nombre de la aplicación que verá el usuario en su teléfono. Normalmente apunta a la cadena "app_name" que se encuentra en el recurso strings.xml. El contenido de este atributo aparecerá en la barra de título de la actividad si la tuviera. También se usará en el lanzador de la aplicación si definimos esta actividad como el punto de entrada de nuestra aplicación. Si no lo definimos, se utilizará el label definido en la etiqueta <application>. Será el título por defecto de las *activities* que definamos dentro de la etiqueta **⟨application⟩**.

- theme: Este atributo apunta al archivo de recursos styles.xml, donde se define la personalización del **estilo visual** de nuestra aplicación.

Dentro de <application> encontraremos expresada la actividad principal que definimos al crear el **proyecto Test**. Usaremos <activity> para representar un nodo tipo actividad.

<activity> tiene dos atributos: name, el cual se refiere a la **clase Java** que hace referencia a esta actividad (Comienza con un punto "."). Y el atributo **label** que hace referencia al texto que se mostrará en la cabecera de la actividad. En este caso es el mismo string "app_name".

Contiene la etiqueta ⟨activity⟩

- android:name=*"string"* → Especifica el nombre de la actividad. En el ejemplo se ha puesto .MainActivity ya que tenemos el nombre del *package* definido como atributo del elemento *manifest.*

- android:screenOrientation= "portrait o landscape" → Define la orientación de la aplicación vertical o apaisado. Si no definimos este atributo, se usará el que el dispositivo tenga predeterminado, incluido el variable en función del acelerómetro. Cuando la pantalla cambia de orientación la actividad se destruye.

- android:configChanges= "opcion" → En Android, el cambio de orientación o la aparición del teclado se considera un cambio de configuración, lo que hace que la pantalla se redibuje para adaptarse a los cambios. En este atributo definimos los eventos que queremos manejar nosotros, con el fin de evitar que se destruya la actividad. Los diferentes efectos se separan con la barra vertical o *Pipe* (|), Podríamos

poner *"keyboard|keyboardHydden|orientation"*, para la aparición desaparición de la pantalla y para el cambio de orientación del dispositivo.

En *Android* no existe un único punto de entrada para nuestra aplicación. Podemos iniciarla a través de múltiples *activities* o *services* que pueden ser iniciados a partir de *intents* específicos que puede enviar el sistema u otra aplicación. Para decir a Android ante qué *intent* debe reaccionar nuestra aplicación y cómo, existe el <intent-filter>.

La Etiqueta ⟨intent-filter⟩

- Con el elemento <action>, estamos indicando que esta *activity* va a ser un punto de entrada para nuestra aplicación. Sólo puede haber una *activity* que reaccione a este *intent.*

- Con elemento <category> le decimos a *Android* que queremos que esta *activity* sea añadida al lanzador de la aplicación. Pueden haber varias *activities* que reaccionen a este *intent.*

De esta manera *Android* sabe que al ser pulsado el icono de la aplicación, debe iniciar esta *activity.* Si no hubiéramos establecido el *<intent-filter>*, esta *activity* sólo se podría llamar desde dentro de la aplicación. Fijémonos que ambos elementos sólo tienen el atributo name siendo su valor el nombre del *intent.*

```
<actionandroid:name="android.intent.action.MAIN"/>
<categoryandroid:name="android.intent.category.LAUNCHE/>
```

Una activity definida sin estos elementos solamente podría ser accedida desde dentro de la aplicación.

Etiqueta uses-permissions

Android cuenta con un modelo de seguridad muy elaborado. Cada aplicación se ejecuta en su propio proceso y "máquina virtual". Android restringe el uso de los recursos del sistema: tarjeta SD, WIFI, HW de audio, etc.

Mediante este Tag especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse, además son los que deberá aceptar el usuario antes de instalarla. Por ejemplo, si se desea utilizar funcionalidades con Internet o el vibrador del teléfono, hay que indicar que nuestra aplicación requiere esos permisos.

Alguno de estos permisos son:

android.permisssion.RECORD_AUDIO para el acceso al hw de audio y grabación.

android.permisssion.INTERNET, concede permiso a todas las API's de red.

android.permisssion.WRITE_EXTERNAL_STORAGE, para almacenamiento externo.

android.permisssion.WAKE_LOCK, nos permite antibloqueo (que si el terminal está inactivo éste no se bloquee).

Etiqueta uses-features

Permite especificar las características hardware que requiere nuestra aplicación (pantalla multitáctil, soporte OpenGL, etc.)

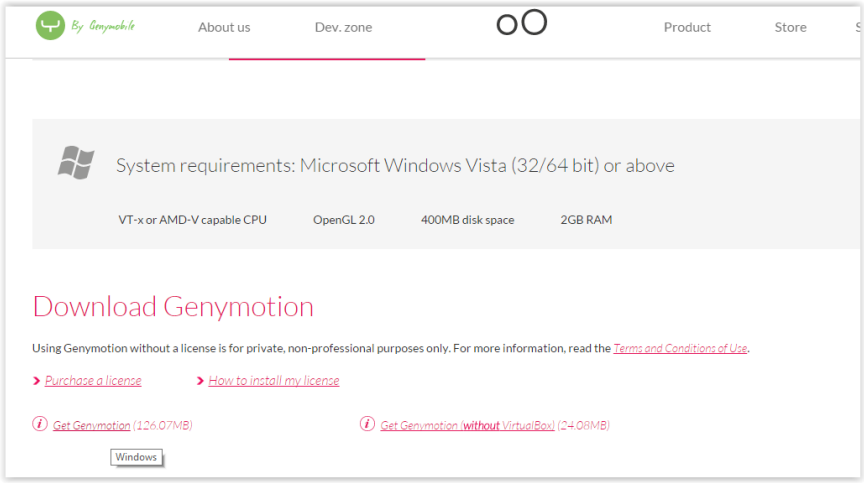
Algunas restricciones hardware para nuestra aplicación podrían ser:

```
<uses-featureandroid:name="android.hardware.microphone" android:required="false"/>
<uses-featureandroid:name="android.hardware.camera" android:required="false"/>
<uses-featureandroid:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-featureandroid:name="android.hardware.camera.flash" android:required="false"/>
<uses-featureandroid:name="android.hardware.touchscreen.multitouch" android:required="true"/>
```

Android Manifest Para profundizar más

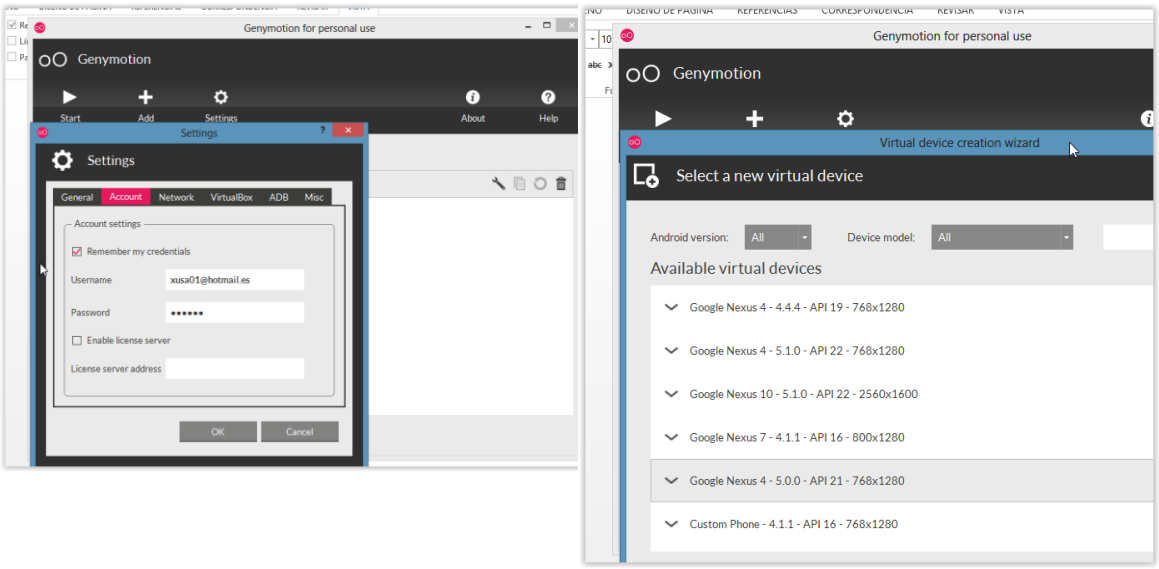
GENYMOTION, EMULADOR DE DISPOSITIVOS

Para poder probar nuestra aplicación podemos usar cualquier dispositivo Android o usar un emulador. Android Studio aporta su propio emulador, pero podremos mejorar el tiempo de respuesta a la hora de la ejecución en el dispositivo, si utilizamos el emulador de Genymotion. Podemos descargar la versión gratuita de la dirección https://www.genymotion.com/#!/download.

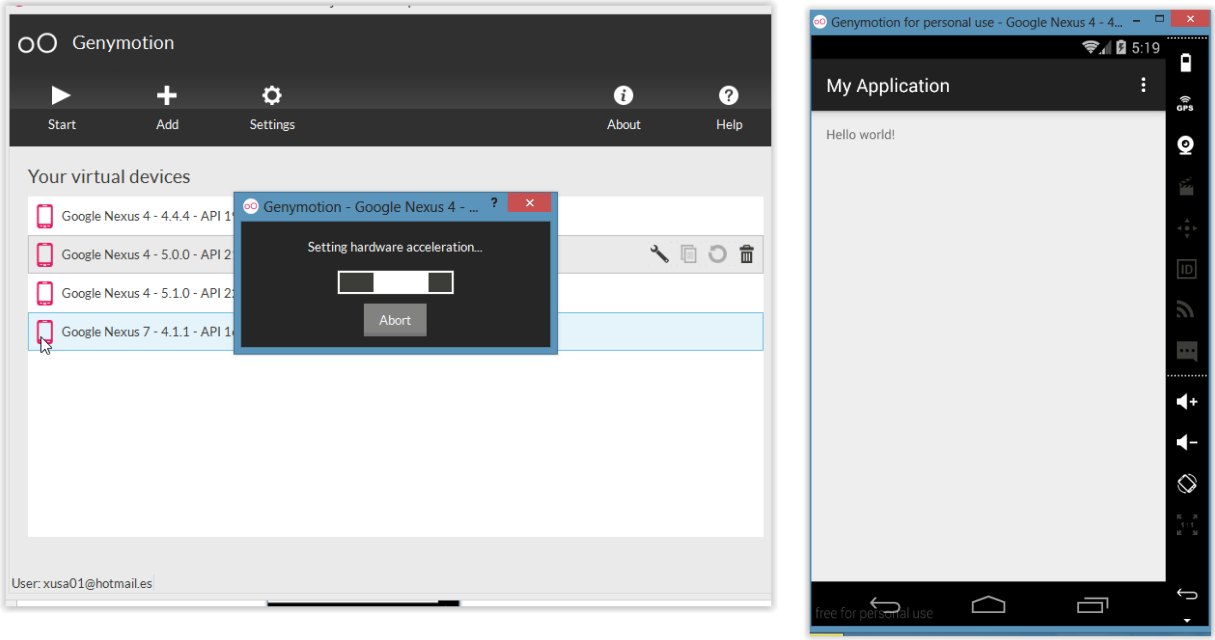


Será importante descargar el archivo que lleva incluido el VirtualBox, si no lo tienes instalado en tu equipo. En la misma página podemos ver las instrucciones de instalación (deberemos crear una cuenta) y como añadir el plugin a Android Studio.

Una vez instalado deberemos descargar las máquinas virtuales que nos interesen, para ello tendremos que configurar nuestra cuenta, **setting ->account**. Y después en **Add** conectaremos con el servidor, donde aparecerán todas las que nos ofrece Genymotion, nosotros iremos instalando las que necesitemos.

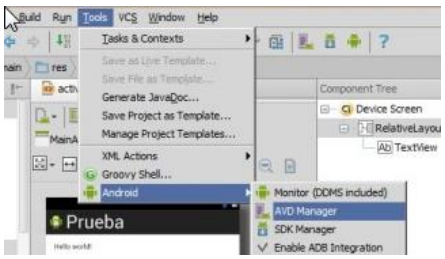


Cuando esté instalada solamente tendremos que seleccionar la que queramos y con Start, esperar un poquito a que arranque (la primera vez tarda más).

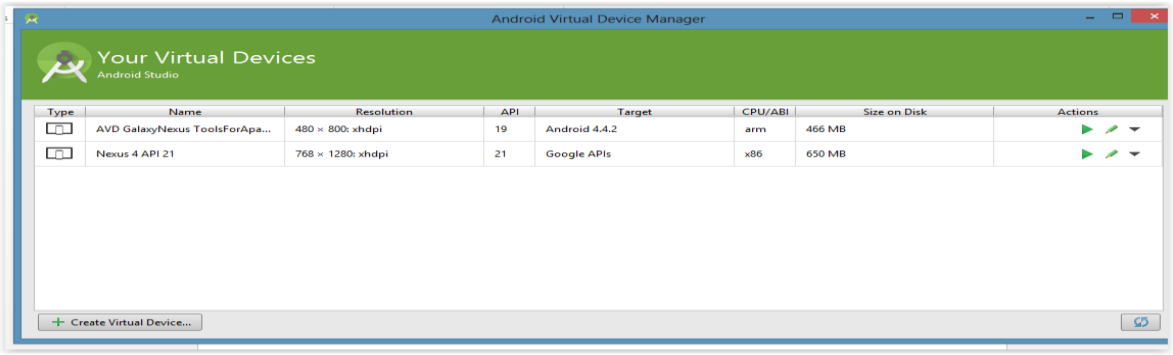


AVD, EL EMULADOR DE SMARTPHONE Y TABLETS ANDROID STUDIO

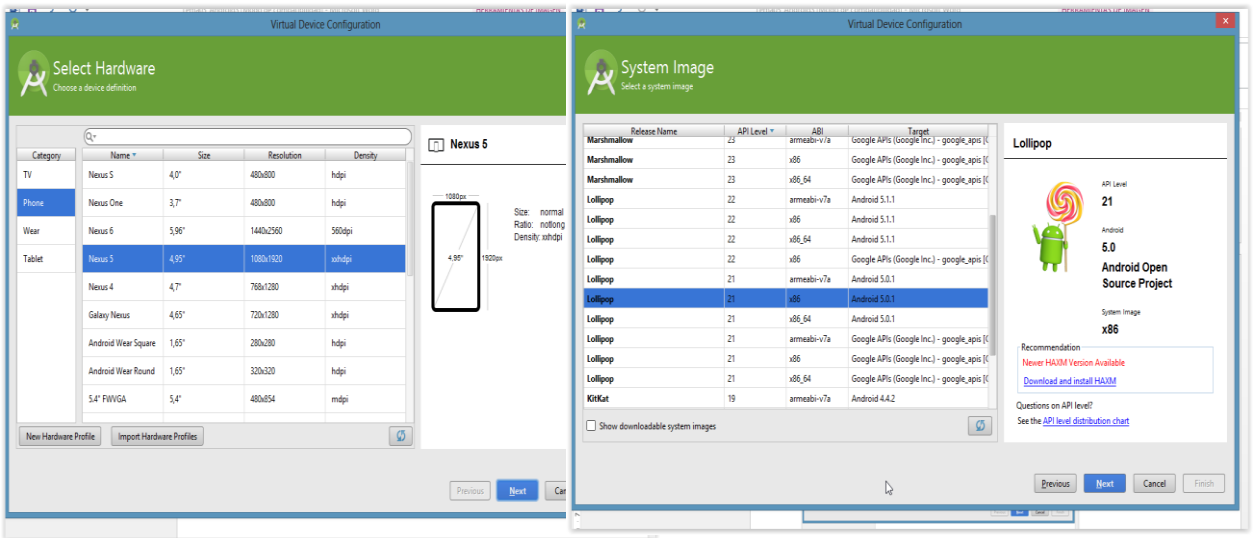
Para probar nuestras aplicaciones Google nos proporciona un **emulador de dispositivos conocido como AVD** (Android Virtual Devices). Pero antes de empezar a usarlo y probar nuestra aplicación debemos generar un dispositivo emulado. Para ello, debemos ir a Tools en el menú superior y seleccionar Android, y dentro **AVD Manager**.



En la ventana que se abrirá debemos pulsar en **Create Virtual Device:**

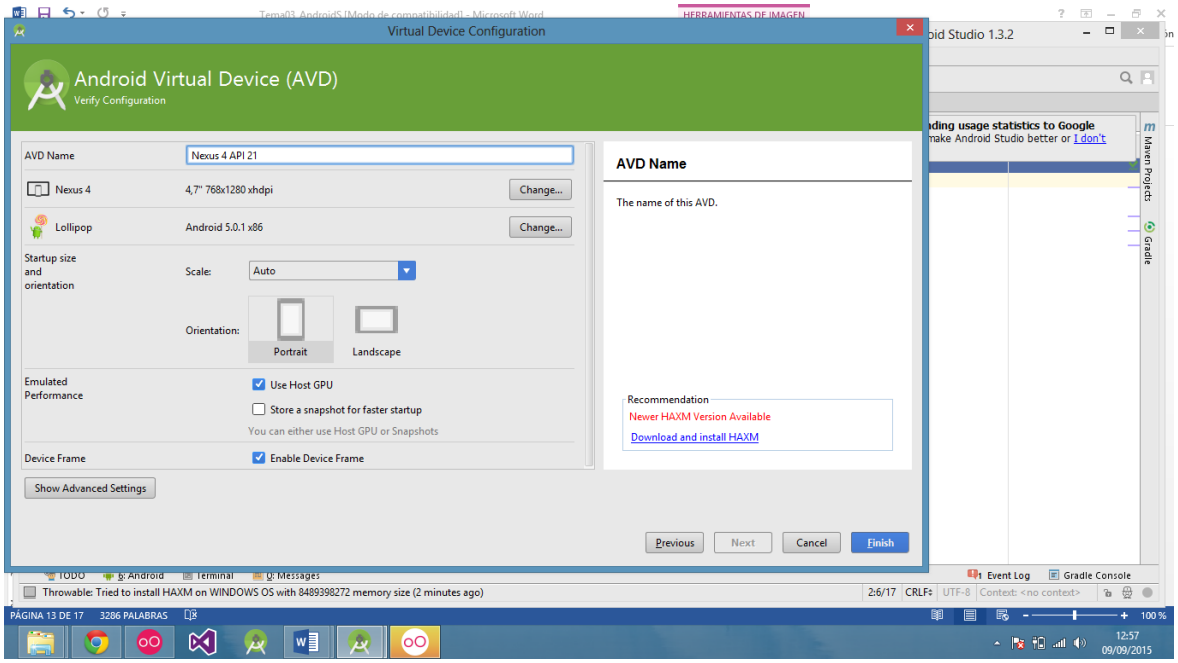


Primero seleccionaremos el dispositivo con el que queremos probar el desarrollo, y después la versión del Sistema Operativo que queremos que tenga este.



La siguiente pantalla mostrará la edición de la máquina virtual que se está creando, y podremos modificar algunas de sus características a nuestra conveniencia. Tendremos que tener en cuenta que si queremos instalar versiones con el procesador Intel, habrá que instalar el Intel x86 Emulator Accelerator (HAXM installer), primero deberemos añadirlo en el SDK (en Tools), y después tendremos que instalar el software necesario, se encuentra en el directorio_dondeestá_intalado_el_SDK_de_Android:

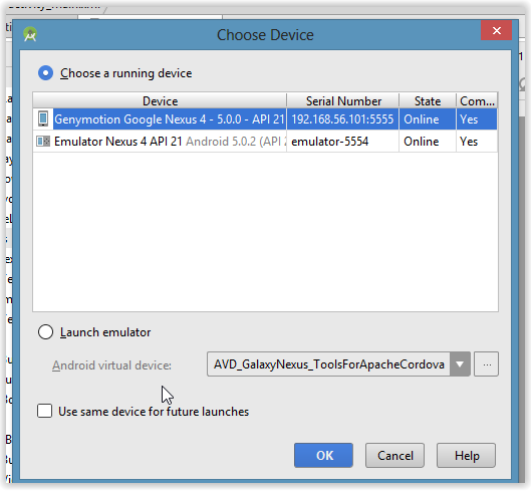
extras\Intel\Hardware_Accelerated_Execution_Manager e instalar el intelhaxm-android.exe (puede que tu procesador no lo soporte, en ese caso deberás utilizar procesador arm.



Aceptamos y volvemos a la ventana anterior. En ella vemos que se nos **ha creado un dispositivo virtual** con el nombre que le hayamos puesto y junto a él, el resto de características. En esta ventana, además de crear podemos editar las características, borrar un dispositivo virtual, ver los detalles en profundidad e iniciar la emulación.

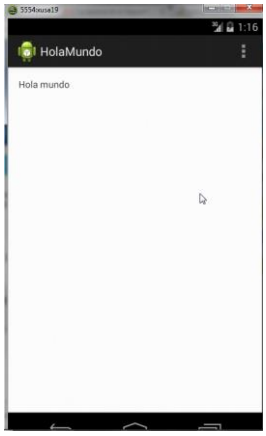
Ejecutar la aplicación

Cuando tengamos un emulador arrancado o un dispositivo Android conectado a nuestro ordenador, podremos ejecutar nuestra aplicación, para ello podremos pulsar mayus+F10 o seleccionar el botón del play. En este paso, el **IDE revisará todo el proyecto y nos avisará si nuestra aplicación tiene algún error o advertencia que se nos haya pasado por alto**. En el caso de ser un error, no mostrará la siguiente ventana, si no mostrará el error.



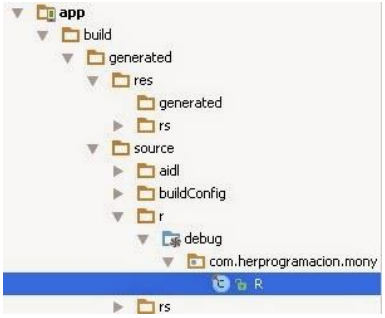
En caso de una advertencia, nos indicara si queremos continuar o solucionarla antes de proseguir. Si por el contrario, lo tenemos todo correcto, veremos una ventana para seleccionar un dispositivo virtual, lo seleccionaremos daremos a OK y a esperar a que arranque el emulador para que se ejecute la aplicación.

Usaremos el ratón como dedo y desplazaremos para desbloquear y ¡Voilá! tendremos nuestra aplicación funcionando. **Un "Hola Mundo en toda regla**.



COMPRENDIENDO EL PROPÓSITO DE LA CLASE R.JAVA

El archivo R.java es una archivo que se autogenera dentro de la carpeta build, para enlazar todos los recursos que tenemos en nuestro proyecto al **código Java**.



Si abres el archivo podremos ver un código similar a este:

```
/*AUTO-GENERATED FILE. DO NOT MODIFY. *
* This class was automatically generated bythe
* aapt tool from the resource data itfound. It
* should not be modified by hand. */
package com.TUPAQUETE.test;

public final class R {
   public static final class attr {
   }
   public static final class dimen {
      public static final int activity_horizontal_margin=0x7f040000;
      publicstaticfinalint activity_vertical_margin=0x7f040001;
   }
   public static final class drawable {
      publicstaticfinalint ic_launcher=0x7f020000;
   }
   public static final class id {
      public static final int action_settings=0x7f080000;
   }
   public static final class layout {
      publicstaticfinalint activity_my=0x7f030000;
   }

public static final class menu {
      publicstaticfinalint my=0x7f070000;
   }
   public static final class string {
      public static final int action_settings=0x7f050000;
      public static final int app_name=0x7f050001;
      public static final int hello_world=0x7f050002;
   }
   public static final class style {
      /** Customize your theme here.
       */
      public static final int AppTheme=0x7f060000;
   }
}
```


Como ves, la clase R contiene clases anidadas que representan todos los recursos de nuestro proyecto. Cada atributo tiene una dirección de memoria asociada referenciada a un recurso específico. Por ejemplo, la clase string posee el atributo hello_world, el cual representa nuestro TextView en la actividad principal. Este recurso está ubicado en la posición0x7f050002.

No modifiques el archivo R.java, el archivo se actualiza automáticamente al añadir un nuevo elemento al proyecto.

ACTIVIDADES

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.xusa.miaplicacion"
   android:versionCode="11"
   android:versionName="1.5.1">

   <application android:icon="@drawable/icon">
      <activity
         android:name=".UnActivity"
         android:configChanges="orientation|keyboardHidden"
         android:label="@string/app_name"
         android:screenOrientation="portrait"
         android:theme="@style/Theme.NoBackground">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity
         android:name="OtroActivity"
         android:configChanges="orientation|keyboardHidden"
         android:label="@string/app_name"
         android:screenOrientation="portrait"
         android:theme="@style/Theme.NoBackground"></activity>

      <!-- Elementos para la publicidad -->
      <meta-data
         android:name="com.mobclix.APPLICATION_ID"
         android:value="insert-your-application-id" />
      <activity android:name="com.mobclix.android.sdk.MobclixBrowserActivity" />
   </application>

   <supports-screens
      android:anyDensity="true"
      android:largeScreens="false"
      android:normalScreens="true"
      android:smallScreens="false"></supports-screens>

   <!-- Permisos que se exigen para mostrar la publicidad -->
   <uses-permission android:name="android.permission.INTERNET"></uses-permission>
   <uses-permission android:name="android.permission.GET_TASKS"></uses-permission>
   <uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
   <!-- Permisos en General -->
   <uses-permission android:name="android.permission.VIBRATE"></uses-permission>
   <!-- Version mínima de androide soportada por la aplicacion -->
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

- **Realiza una descripción de la aplicación a través del análisis del archivo de manifiesto anterior.**
- **Modifica el Manifest del ejercicio HolaMundo de modo que como requisitos del mismo queremos que se ejecute en modo apaisado y que no afecte el cambio de orientación del dispositivo. Cambia el icono y la etiqueta de la activity principal (probar que si no la tiene pone por defecto la de la aplicación) y dale permisos de internet.**